SUPLEMENTO MENSUAL DEDICADO A LA IMAGEN SINTÉTICA

RENDERMANÍA

Número 6

La Portada
Programación escénica de desastres virtuales

Cómo...
Preparar el acabado de superficies en Imagine

Taller Virtual
La atmósfera en POV 3.0

Foro del lector
Piezas para juegos y batallas virtuales

pcmania

José Manuel Muñoz

Preparar el acabado de

El número anterior fue el segundo dedicado a Imagine. Hasta ahora hemos estudiado los Editores de Formas, Detalles, Escenas y Proyectos. En conjunto lo ya visto es suficiente para permitir al lector acometer proyectos de cierta envergadura, aunque quedan muchas cosas por tratar. Por ejemplo, ¿cómo crear acabados planos en los objetos?, ¿cómo aplicar bitmaps?...

El acabado es uno de los apartados más importantes al generar imágenes sintéticas. Empezaremos viendo cómo crear acabados planos o redondeados. Luego repasaremos el Gestor de Atributos y concluiremos con el tema de la aplicación de bitmaps y la generación de sombras.

Superficies redondeadas y alisadas

Existen muchos algoritmos de sombreado de superficies. Imagine, por defecto, emplea el sombreado Phong con el cual los objetos toman una apariencia redondeada ante nuestra vista, a pesar de que, realmente, están construidos mediante una malla de caras planas o facetadas. Con este algoritmo se calcula el

Todo los ficheros de inclusión y los ejemplos podéis encontrarlos en el directorio PCMANIA\RENDER56 del CD-Rom.
La versión completa de Imagine 3.0, junto con las texturas, se incluye, de nuevo, en el CD-Rom de este número.

superficies en Imagine

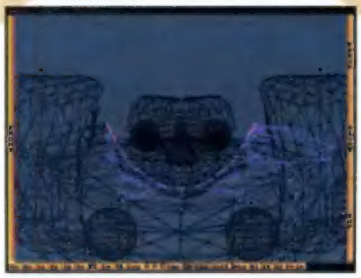
color adecuado para cada punto de la superficie atendiendo a la inclinación de ésta con respecto a la fuente de luz. Este sistema -conocido ya por la mayoría de los lectores- permite a los diseñadores 3d crear objetos de apariencia suavizada y realista empleando pocos polígonos. El sombreado Phong requiere un tiempo de cálculo muy superior al llamado flat, donde sólo se calcula un único color para todos los puntos de un mismo polígono, pero también es mucho más real.

Como ya vimos en el número anterior, podemos ordenar diferentes tipos de sombreado desde la ventana de subproyectos. De esta manera todos los objetos se verán con una apariencia facetada (con "Color shaded") o bien suavizada (con "Scanline" o "Trace"). Lo normal es exigir el modo de máxima calidad ("Trace").

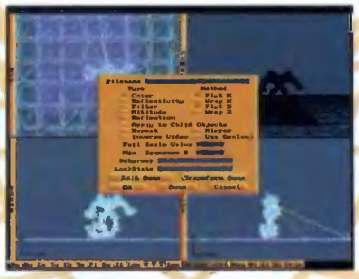
Comentamos todo esto porque hay un problema que se presentará con frecuencia en Imagine aunque ordenemos la máxima calidad. Existen objetos cuyas superficies son total o parcialmente planas. En estos casos el programa tendrá que realizar un sombreado liso o suavizado según sea el caso. Algunos programas toman nota de cuando deben aplicar el suavizado o no realizando cálculos de ángulos entre polígonos (si el ángulo es menor de X, se efectúa el suavizado). Pues bien, Imagine no es uno de ellos. Imagine aplica siempre, por defecto, un sombreado suavizado en todas las caras y ello puede hacernos quedar perplejos al observar el acabado final de muchos objetos (como cubos) que sabemos de superficie lisa. No obstante, ya hemos visto escenas de Imagine con objetos que tienen zonas lisas y zonas redondeadas. Sin ir más lejos, la cabeza de Ray (el muñeco pequeño de las gafas de los números 4 y 5) presenta zonas de las dos clases. ¿Cómo se hizo esto?

Veamos; ante todo hay que indicar a Imagine qué partes del objeto deberán mostrarse alisadas y cuáles redondeadas. En primer lugar, desde el Editor de Detalles, seleccionaremos el objeto a tratar pinchando sobre él (en el modo de objetos o en el de grupos). Una vez seleccionado éste cambiaremos al modo "Pick Edges", en el submenú "Mode". Al hacerlo observaremos como los vértices del objeto quedan recuadrados en amarillo. Entonces -desde la ventana apropiada- dibujaremos una caja de selección ("Drag Box" en el apartado "Pick Method" del submenú "Mode") mientras mantenemos pulsada la tecla de Shift (modo múltiple). Los puntos contenidos por la caja quedarán seleccionados y podremos añadir nuevos puntos a esta selección usando nuevas cajas. Hecho esto iremos al submenú "Fuctions" y pincharemos en "Make". Aparecerá otra ventana de la que destacaremos dos opciones; "Make sharp edges" y "Make soft edges". Si escogemos la primera, las caras implicadas presentarán una apariencia lisa. Si escogemos la segunda, la apariencia será suavizada. Normalmente, como por defecto toda la superficie de los objetos está suavizada, escogeremos la primera opción.

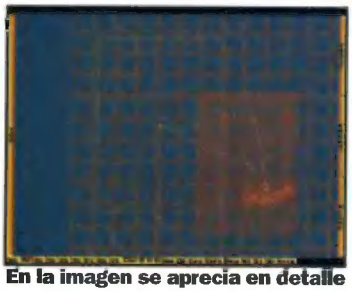
Ir seleccionando puntos aunque sea usando una caja de arrastre puede ser engorroso, sobre todo si el objeto es complejo, y por ello Imagine dispone de un segundo método para efectuar estas selecciones. En el submenú "Pick/Select" hay una opción llamada "Edge Filter" (Filtro de aristas). Al pinchar sobre ella aparecerá una ventana mediante la



Momento de la selección de bordes para el suavizado.

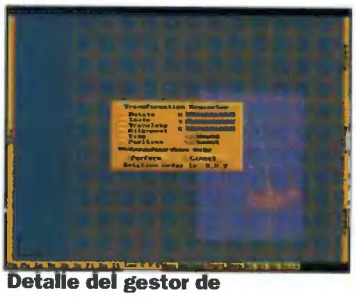


La ventana de tipos de aplicación de bitmaps.



En la imagen se aprecia en detalle la aplicación de un bitmap.

El gestor de las fuentes de luz es vital para lograr un buen efecto.



Detalle del gestor de transformaciones para los ejes.

cual podremos ordenar una selección de aristas -que cumpla ciertas condiciones- en el objeto o grupo seleccionado. Las condiciones más importantes son el ángulo mínimo y máximo entre los que debe estar comprendida la arista. Aparte de esto podemos ordenar que la selección se lleve a cabo sólo entre las aristas lisas ("Sharp edges only") o entre las suavizadas ("Soft edges only"). En cuanto pulsemos sobre el botón de aceptación, Imagine tomará nota del modo en que deberán visualizarse las aristas en el proceso de Render.

El gestor de atributos

Como ya comentamos en el número anterior, para crear los atributos (características de superficie) de un objeto o modelo seleccionaremos el Editor de Detalles (desde el modo de objetos o grupos, según sea el caso), y pulsaremos sobre "Attributes Requester", en el submenú "Functions". Entonces aparecerá la ventana del Gestor de Atributos donde se hallan los botones para dar color, transparencia (filter), rugosidad (roughness) y otras características a los objetos. En el número anterior ya realizamos una breve descripción de la función de estos botones y no vamos a entrar ahora en más detalles, ya que lo mejor es que el lector experimente con ellos.

Los atributos de superficie así creados se guardan automáticamente con los objetos, cuando estos se graban. Aparte de esto Imagine nos permite crear librerías de atributos separadamente, empleando los botones "Load" y "Save". (Normalmente guardaremos estas librerías en el directorio Attribs).

Aplicación de bitmaps

También es en el Gestor de Atributos desde donde se realiza la aplicación de las texturas. Imagine contempla dos tipos de ellas. Por un lado están las texturas tipo bitmap (a las que Imagine llama "Brush") y por otro están las llamadas texturas procedurales, las cuales se generan siguiendo procedimientos algorítmicos. Hoy nos centraremos en la aplicación de bitmaps o mapas de imagen.

Una vez seleccionado un objeto, y ya en la ventana del Gestor de Atributos, veremos que el recuadro más grande dentro de dicha ventana es el que se halla bajo el texto "Textures/Brushes". En este recuadro figurará la relación de texturas de uno u otro tipo que se vayan a aplicar sobre el objeto (si las hay). Bajo el recuadro hay cuatro botones relacionados con la aplicación de texturas. Por el momento sólo nos interesa el de aplicación de bitmaps ("Add Brsh"). Al pinchar sobre este botón aparecerá una nueva ventana que nos servirá -buscando entre los directorios, si es preciso- para elegir el bitmap adecuado. Entonces, si escogemos un fichero, aparecerá otra ventana desde la que controlaremos la aplicación de la textura. Veamos lo que hacen algunos de sus botones: "Color" es la forma de aplicación por defecto y en ella el color de cada punto del bitmap reemplaza al propio color del objeto. "Filter" utiliza la escala de grises del bitmap para dar diversos grados de transparencia al objeto sobre el que se aplica la imagen. (Las áreas negras son opacas, las blancas transparentes y las grises translúcidas). En cuanto al botón "Altitude", emplea el bitmap para simular irregularidades (bumpings) en la superficie del objeto. Esta opción no produce cambios en la geometría sino que emplea la información de color del bitmap para producir apariencias de abultamientos o depresiones en el objeto.

Otros botones importantes son "Flat X", "Wrap X", "Flat Z" y "Wrap Z". Con ellos especificaremos el tipo de proyección que efectuará el bitmap sobre el objeto, debiendo escoger entre uno u otro tipo según la forma del objeto. En la portada del número anterior vimos un ejemplo de proyección plana (flat). En ella teníamos un bitmap que queríamos aplicar sobre un objeto de forma alisada para simular una revista. En la proyección plana, los puntos del bitmap se aplican perpendicularmente al plano formado por la superficie del objeto (en este caso la revista) a fin de evitar distorsiones visuales. Estas distorsiones son visibles como "estiramientos" del bitmap en las zonas del objeto donde la

superficie deja de ser perpendicular al eje de aplicación. De hecho, si uno de los polígonos del objeto queda paralelo al eje de aplicación, veremos que cada punto del bitmap se estira indefinidamente para cubrir cada punto de la superficie del mismo. Para comprender esto imaginemos un cubo sobre el que realizamos una aplicación plana perpendicular a una de sus caras. La aplicación resultará perfecta en dicha cara y también en la otra cara paralela del cubo pero en las otras, en cambio, apreciaremos el feo efecto de estiramiento. Esto se debe a que en la aplicación plana del bitmap, cada pixel del mismo atraviesa espacialmente al objeto en la dirección del eje de aplicación y se deposita en cada punto de la superficie, al entrar en el objeto o salir de él. Para evitar este efecto hay dos sistemas: uno cambiar la orientación del eje de aplicación de modo que no sea paralelo a ninguna de las caras del objeto (lo cual es una chapuza) y otro es cambiar el propio objeto. Esto último fue lo que se hizo en la portada del número anterior, donde el bitmap se aplicó sólo en un plano puesto encima del objeto que hacía las veces de revista. El modo de aplicación plano es el empleado por defecto por Imagine.

En la aplicación esférica, en cambio, el bitmap se enrolla alrededor del objeto siguiendo la forma del mismo. Usaremos los 4 botones antes mencionados como switches cuando deseemos realizar una aplicación que resulte plana en un eje y esférica en otro. (Para un cubo marcaríamos "Flat X" y "Flat Y". Para un cilindro marcaríamos Wrap en un eje y Flat en otro y para una esfera marcaríamos Wrap en los dos).

Otros botón de la ventana con gran importancia es "Edit Axes", con el cual podremos controlar directamente la aplicación del bitmap. Supongamos que hemos elegido el modo flat. Al pulsar sobre "Edit Axes" veremos como se visualiza un plano (en una de las ventanas) sobre el objeto en ha de efectuarse la aplicación. Este plano nos está indicando la zona donde va a aplicarse el bitmap y su orientación y escala con respecto al objeto. Podremos alterar estas características usando los conocidos botones de escalación, rotación y desplazamiento de la fila inferior de botones. Una vez que quedemos contentos con la aplicación, pulsaremos espacio y retornaremos a la ventana de aplicación, donde pulsaremos sobre "Ok". Entonces volveremos a la ventana anterior donde veremos como el nombre del bitmap se ha incluido en el recuadro "Textures/Brushes".

Para eliminar este bitmap o cambiar la aplicación del mismo sobre el objeto pincharemos sobre "Info", lo cual nos hará entrar nuevamente en la ventana de aplicación desde la cual podremos llevar a cabo la modificación o bien eliminar la aplicación pinchando en el botón "Drop".

Modo flat.

Acabado suavizado.

Añadiendo texturas.

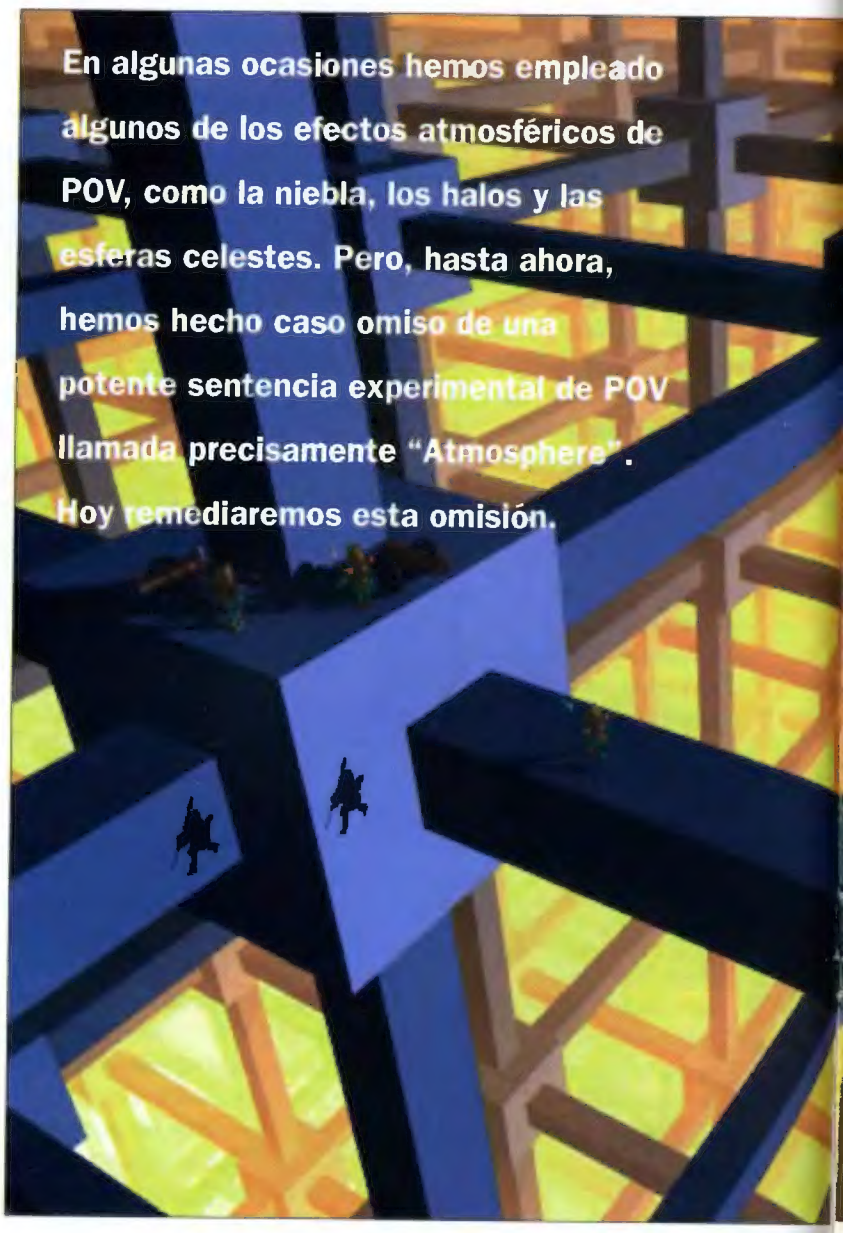
Añadiendo sombras.

Sombras

Para que los objetos arrojen sombras habremos de activar el flag "Cast Shadows" en la ventana "Light Source Info". Para acceder a esta ventana tendremos que ir al Editor de Acciones, cuyo funcionamiento no vamos, por ahora, a explicar. Para nuestros fines bastara con saber que es aquí donde podemos cambiar las propiedades de la fuente de luz. Al entrar en el Editor veremos una pantalla de trabajo donde cada objeto tiene un nombre situado en alguno de los recuadros de la izquierda. Buscaremos el nombre de la fuente de luz (normalmente "Lightsource") y pincharemos primero en el botón "Info", situado en la esquina inferior izquierda, y luego en uno de los puntos de color del cuadro situado a la derecha del nombre de la fuente. Entonces aparecerá la ventana "Light Source Info", donde marcaremos el recuadro antes mencionado.

La atmósfera en POV 3.0

Ante todo conviene empezar repitiendo la advertencia del Povteam: la sentencia "Atmosphere" es aún experimental y existe una alta probabilidad de que sea modificada en una futura versión de POV, de modo que se pierda la compatibilidad con la instrucción actual. De todos modos, teniendo en cuenta el interés intrínseco de esta sentencia, hemos decidido trastear con ella.



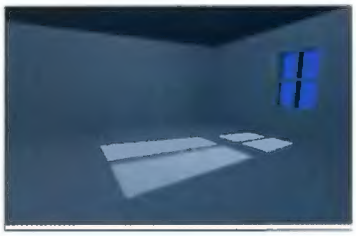
En algunas ocasiones hemos empleado algunos de los efectos atmosféricos de POV, como la niebla, los halos y las esferas celestes. Pero, hasta ahora, hemos hecho caso omiso de una potente sentencia experimental de POV llamada precisamente "Atmosphere". Hoy remediaremos esta omisión.

El modelo atmosférico de POV

Según el manual de POV: "la sentencia atmosphere se emplea para reflejar la interacción de la luz con las partículas del aire". Con Atmosphere, las emisiones de luz serán visibles y los objetos arrojarán sombras sobre la niebla o el polvo suspendido de la atmósfera. Esto es un paso adelante con respecto a la sentencia Fog, la cual no es sino una aproximación bastante simple al fenómeno de la niebla (recordemos que Fog no interactúa con la luz). Por supuesto, Atmosphere es también, como cualquier otro intento de simular los fenómenos del mundo real en un ordenador, una aproximación. Pero produce resultados más reales que Fog gracias al muestreo de volúmenes espaciales que realiza su algoritmo para calcular la interacción entre la luz y las partículas atmosféricas.

El actual modelo atmosférico generado con Atmosphere asume una den-

El cuarto sin atmosfera y la atmosfera con un valor bajo de muestreo



sidad constante de partículas en el aire, exceptuando el volumen espacial de los objetos sólidos. Si queremos crear nubes, humo u otras distribuciones irregulares de partículas deberemos recurrir a los halos. La sintaxis de la sentencia es:

```
atmosphere {
  type NUMERO_DEL TIPO
  distance DISTANCIA
  [ scattering SCATTERING ]
  [ eccentricity ECCENTRICITY ]
  [ samples SAMPLES ]
  [ jitter JITTER ]
  [ aa_threshold AA_THRESHOLD ]
  [ aa_level AA_LEVEL ]
  [ colour <COLOUR> ]
}
```

La palabra "Type" se usa para indicar el tipo de modelo de dispersión luminosa que va a emplearse (o sea, cómo van a dispersar la luz las partículas de la atmósfera). El modelo más sencillo y rápido es el isotrópico (el 1), en el cual la cantidad de luz dispersada no depende, como en los otros modelos, del ángulo entre la dirección de visión y el rayo de luz. El modelo isotrópico es el que hemos empleado en todos los ejemplos.

La siguiente palabra, "Distance", es usada por POV para averiguar la densidad deseada de las partículas de la atmósfera. Como antes se ha dicho, esta densidad será constante para toda la atmósfera. "Distance" funciona aquí igual que para la niebla constante en la que, recordemos, cuando la distancia de un objeto iguala a la del valor dado a "Distance", el color del objeto estará compuesto por un 64% de color propio y por un 36% del color de la niebla.

La palabra "Scattering" se usa para cambiar la cantidad de luz que es dispersada por la atmósfera. De esta manera controlaremos el brillo de la misma. Los valores pequeños disminuyen el brillo y los valores altos lo incrementan. La luz que no es dispersada, es absorbida.

Con la palabra "Color" (o "Colour") podremos especificar un color para la atmósfera. El color puesto por defecto es el negro. En cuanto a la cantidad de luz que es filtrada por el color de la atmósfera, depende del valor dado en el parámetro filtro. Por ejemplo con "rgbf <1,0,0,0.40> ", el 40% de la luz será filtrada hacia el color de la atmósfera. Así pues, un valor de .10 significa un tono muy tenue de rojo y un valor de .70 implica una atmósfera "marciana". También podemos emplear "Transmittance" en vez de "filter" (o sea; "rgbt"). Transmittance, que funciona aquí igual que en Fog, se emplea para especificar el mínimo de translucidez de la atmósfera.

El algoritmo empleado por "Atmosphere" funciona realizando muestreos del volumen espacial a lo largo del rayo de visión. En cada punto muestreado se tomará nota de si dicho punto cae dentro de un rayo de luz o no. Si un punto muestreado cae dentro de una zona iluminada, la cantidad de luz dispersada en la dirección del observador será determinada y sumada a la intensidad to-

La atmósfera con supermuestreo.

La atmósfera con color negro.

La atmósfera con un filtro de rojo.

tal. El número de estos muestreos será determinado por la palabra "samples", que indicará el número de muestreos que se efectuarán para cada intervalo en el rayo de visión. La longitud de cada intervalo será o bien la distancia dada por "Distance" o bien la longitud hasta una parte iluminada por un rayo de luz. En cualquier caso lo que debemos recordar es que a mayor número de muestreos, más reales serán los resultados.

Otro punto a tener en cuenta es que, como el resultado final depende de una serie de testeos a lo largo de un volumen espacial, será conveniente realizar un antialias en los resultados. Por supuesto, podemos limitarnos a dar un valor lo suficientemente alto a "samples", pero esto sería parecido a generar siempre imágenes a resoluciones muy altas (en cuanto a coste de tiempo). Otra solución viene dada por las palabras "Jitter", "aa_level" y "aa_threshold".

Usando jitter, el punto espacial donde va a efectuarse el muestreo es desplazado una pequeña distancia aleatoria a lo largo de la dirección del rayo del observador, lo cual ayuda a reducir el problema del "escalonamiento". Un buen valor para jitter es 0.5. Valores más altos pueden producir un exceso de "ruido" visual en los resultados.

Otra posibilidad para evitar este problema es efectuar un "super-muestreo" (super-sampling). Esta técnica realiza muestreos adicionales en las zonas donde las intensidades entre dos puntos muestreados son muy diferentes. En ese caso se realizarán más muestreos de forma recursiva hasta que se alcance el nivel de recursión indicado o hasta que las diferencias de intensidades entre los puntos muestreados queden dentro de lo tolerable. La palabra "aa_level" indica el nivel máximo de recursión y la palabra "aa_threshold", la diferencia máxima permitida entre la intensidad de dos muestreos (a fin de dar por acabado el proceso).

Los ejemplos de POV

En el directorio docsdemo de POV se incluyen varios ficheros de ejemplo cuyo nombre empieza por atmos para ilustrar el uso de "atmosphere". En todos ellos (salvo en el último) se muestra una habitación iluminada con una fuente puntual. La habitación consta de una ventana por la que entra la luz de una fuente spotlight, que ilumina el suelo. En el primer ejemplo, atmos1, no hay atmósfera de ninguna clase. En el siguiente caso, atmos2, se añade la atmósfera pero el resultado, un efecto de escalonamiento luminoso, no es muy bueno. En el tercer ejemplo hay algunas mejoras que dan un resultado más vistoso. La primera mejora consiste en un valor de muestreo bastante mayor. Además, se añade un porcentaje de "jitter" y se añade el supersampleado.

Cuestiones a tener en cuenta

Cuando usemos "Atmosphere" debemos asegurarnos de que todos los objetos que deban tener atmósfera sean declarados como huecos con hollow. Esto tendremos que hacerlo incluso con la primitiva "plane", tal como sucedía con los halos.

De igual manera, "Atmosphere" no funcionará si la cámara no está colocada en un objeto hollow.

Cuando, después de realizar una serie de pruebas, deseemos lanzar una imagen definitiva a mayor resolución, tendremos que aumentar el número de muestreos. De lo contrario apreciaremos efectos de escalonamiento que no resultaban visibles a resoluciones inferiores.

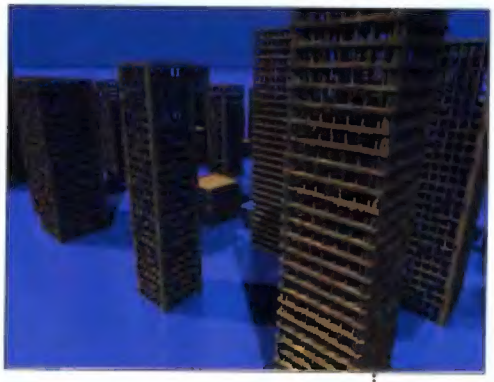
Por defecto, las fuentes de luz interactúan siempre con la atmósfera. Si en algún caso deseamos evitar esto bastará con añadir la palabra "atmosphere off" en la declaración de la fuente. Esto es lo que hemos hecho con la fuente puntual azul de la escena de la malla tridimensional. La otra fuente de luz es una "Spotlight" corriente con "Atmospheric_attenuation on" (en la escena de la malla 3d se ha sustituido la antigua niebla que se puso en el número 2, por una sentencia "atmosphere"; este experimento, no demasiado logrado, fue el primero realizado por el autor con esta sentencia).

Normalmente los rayos de una fuente no se debilitan -en POV- al atravesar la niebla o la atmósfera. Si deseamos cambiar este comportamiento usaremos la palabra "Atmospheric attenuation" dentro de la declaración de la fuente. Este atenuamiento sólo funcionará si la escena tiene atmósfera o niebla.

Programación de un desastre a escala global

Quienes dominan la filosofía de POV saben que las características de su lenguaje escénico -sobre todo las de sus sentencias "de programación"- permiten la creación de escenas que resultarían sumamente difíciles con otros programas. Esto es algo que ya hemos comprobado en diversos números de Rendermanía, pero aún no puede decirse que hayamos explorado todas las posibilidades de la "pov-programación" como herramienta de diseño infográfico. ¿Existen las funciones de usuario en POV? ¿Qué es la recursión? ¿Cómo pueden emplearse las sentencias de POV para crear edificios en ruinas?

Un edificio en ruinas o -peor aún- una ciudad en ruinas viene a ser el típico ejemplo de proyecto que resulta muy fácil de realizar con POV pero bastante difícil de acometer con otros programas como Imagine o 3D Studio. La razón es, por supuesto, que en el diseño de un proyecto de este tipo intervienen factores repetitivos o patrones que son fácilmente representables si disponemos de un lenguaje escénico (como el de POV). Con un lenguaje podremos crear bucles, cambios en el diseño básico dependiendo de ciertas condiciones, y otras cosas. Hay que tener en cuenta que un edificio corriente (en ruinas o no) equivale a un montón de objetos simples que se repiten una y otra vez tales como vigas, plantas, columnas, etc. Con POV podemos emplar las sentencias de bucles para, con unas cuantas líneas, crear un edificio de este tipo. También podemos emplear las sentencias de azar para introducir pequeñas irregularidades en el tamaño o en la composición de los modelos y -por último- podemos emplear nuevamente las sentencias iterativas para componer una matriz de edificios, creando así una ciudad. Con POV todo



esto puede ser posible con un corto fichero de texto. ¿Os imagináis el trabajo que implicaría el montaje de cada uno de estos elementos "a mano" con un modelador de ventanas?

Lo que pretendemos

La idea de la que nació el proyecto de la ciudad arrasada partió de las barrabasadas a las que sometió Katsuhiro Otomo a Neo-Tokio en su ultrafamoso manga Akira. En dicho manga esta desdichada ciudad es primero destrozada por una bomba nuclear, luego, ya reconstruida, es vuelta a destrozar por Akira, después es sometida a la guerra civil psiquica, más tarde es bombardeada por los americanos y, por último, es vuelta a "ultradestrozar" por Akira. Ni que decir tiene que los fondos de las viñetas de esta obra están llenas de edificios destrozados. El resultado gráfico es espectacular pero, sin embargo, una de las cosas que más saltarán a la vista de cualquier pov-adepto -sobre todo en las últimas páginas- es que casi todo el efecto se logra mezclando edificios que -aunque muy detallados a veces- son muy fáciles de reproducir con POV. Esto se debe a que los edificios futuristas dibujados por Otomo están creados con muy pocos elementos diferentes. Estos elementos, en cada edificio, están desalineados, rotos y retorcidos pero son, en definitiva, de muy pocos clases distintas y de formas muy sencillas. Para comprobar esto crearemos primero un edificio sencillo y luego lo destrozaremos.

Aquí no hay calles (ni playa)

Bueno, realmente si había algo en Akira que era de muy difícil representación; las montañas de escombros que llenaban las calles. Ciertamente habríamos podido representar estos escombros de varias formas distintas, pero la única manera convincente hubiera sido apilar

cientos de pequeños objetos de formas variadas. Además tendríamos que haber representando las irregularidades de las calles; el suelo levantado, las aceras, las farolas, etc. Y además, habría que haber preparado también las entradas de los edificios, lo cual hubiera sido, sin duda, la parte mas difícil del diseño.

Aquí hemos solucionado el problema a la manera gordiana clásica: suprimiéndolo. Esto se ha hecho cambiando el tipo de desastre que en nuestras páginas ha pasado a ser un recalentamiento de la temperatura mundial global, lo cual ha producido el deshielo de los casquetes polares acompañado de la consiguiente subida del nivel del mar. De esta manera, al suponerse que los edificios están parcialmente sumergidos, nos hemos ahorrado todo los problemas antes descritos.

Año 2034

La verdad es que puede decirse que nuestros edificios pertenecen a tres épocas diferentes. En la primera la catástrofe es aún muy reciente y los edificios están prácticamente intactos, en la segunda el tiempo ha pasado y los edificios muestran señales de erosión, y en la tercera ha pasado aún más tiempo –quizá siglos– y los edificios están a punto de derrumbarse. Para representar la primera época usaremos el fichero edifi0.pvm del que eliminaremos las sentencias rand. Además sustituiremos las texturas bitmap en el fichero pov por colores planos. Para la segunda epoca emplearemos el mismo fichero .pvm, sin cambios, y sin sustituir las texturas en el fichero pov. Finalmente, para la tercera época, emplearemos el archivo edifi1.pvm. (A partir de ahora citaremos un año para referirnos a los edificios de cada época; 2034 para los edificios de la primera época, 2056 para los de la segunda y 2110 para los de la tercera).

Para los edificios de cada epoca el modo de construcción es casi el mismo. Cada edificio está compuesto por 4 fachadas, los techos de cada piso, el tejado y su barandilla y las columnas laterales. Cada fachada, a su vez, está compuesta por tres elementos; los bloques horizontales, los verticales y los cristales (si los hay) de las ventanas. Todos nuestros edificios se construyen empleando el mismo algoritmo de construcción, en el cual se suministra al fichero "macro" el número de pisos (npisos) y el número de ventanas (nventanas). Otros parámetros servirán para determinar la forma del edificio. Con las variables "npisos" y "nventanas" el algóritmo del fichero .pvm empleado construye 4 fachadas cuyo tamaño puede determinarse así

ancho=nventanas*4

alto=npisos*5

suponiéndose que cada unidad equivale a un metro.

Examinando los ficheros el lector notará que en cada edificio las fachadas podrían haberse construido empleando unas pocas vigas verticales cuya altura fuese igual a npisos*5. También podríamos haber procedido de manera similar con las vigas horizontales. Pero, en lugar de esto, hemos empleado muchas vigas más pequeñas de tal forma que tendremos que, para cada fachada

numero_de_vigas_verticales= npisos*nventanas

siendo cada viga vertical de 5 metros de altura.

Y lo propio sucede con las vigas horizontales. Quizá el lector se sienta algo perplejo al considerar que podríamos haber logrado el mismo resultado gráfico con muchas menos vigas de mayor longitud. Esto es cierto pero solamente para los edificios del 2034. Para los posteriores nos resultará más conveniente el formato empleado.

Año 2056

Los edificios del 2034 están prácticamente nuevos y resultan decepcionantes puesto que son excesivamente sencillos. Colocados en gran número a cierta distancia de la cámara podrían, quizá, cumplir su papel pero hay que recordar que no tienen puertas, ni adornos de ninguna clase y además resultan excesivamente parecidos y uniformes entre sí.

Los del año 2056, en cambio, ofre-

cen un resultado gráfico más atractivo precisamente porque están más trabajados por el tiempo. Para empezar hemos aplicado una textura creada con Photoshop en la que –aparte de ciertas rugosidades debidas a la erosión– se aprecian grietas y roturas varias. El siguiente paso ha sido aplicar un efecto aleatorio comprendido entre 0 y 12 grados a la orientación de cada una de las piezas-viga que componen las fachadas. También hemos hecho que el sentido del giro para cada pieza dependa de un factor aleatorio. Esto se logra con las líneas

```
#declare a=rand(semilla)
#if (a<.50) #declare a=1
#else #declare a=-1
#end
```

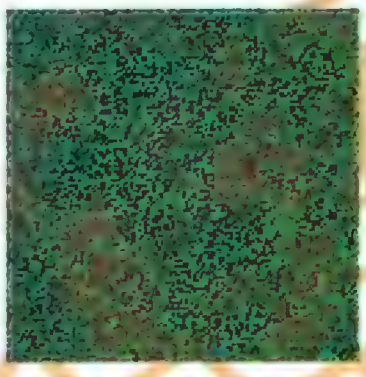
Como recordaréis, la función rand de POV devuelve un resultado flotante aleatorio entre 0 y 1 empleando la semilla inicializada con la función seed. Por tanto las probabilidades de obtener un valor 1 para la variable "a" son del 50%. En las líneas siguientes empleamos "a" para determinar el sentido del giro en cada pieza vertical así, la línea siguiente

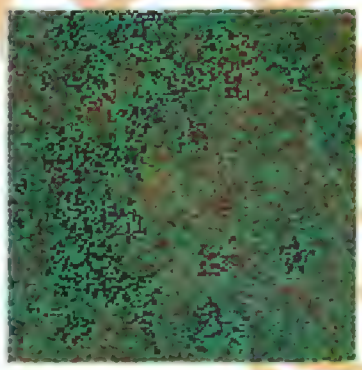
```
object{viga_v rotate x*(rand(semilla)*12)*a}
```

crea una viga vertical a la se que rota una cantidad aleatoria de grados (comprendida entre 0 y 12) en el eje X. El giro se efectuará en un sentido o en otro dependiendo del valor de "a". Este truco también se emplea con las piezas horizontales y con las columnas laterales del edificio cuando las hay.

Año 2110

Aunque el listado es casi el mismo, los edificios del 2056 parecen más conseguidos que los del 2034. Sin embargo aún no podemos ejecutar la danza de la victoria ya que el acabado "catastrófico" del 2056 aún no es perfecto. Como sabemos, en los edificios lo suficientemente "arruinados" suele haber agujeros y roturas que se aprecian en zonas aleatorias de los mismos. Para conseguirlas había que realizar restas espaciales (differences) entre el volumen de cada edificio y otros objetos. Ahora bien, ¿qué tipo de objetos se pueden emplear para efectuar dichas restas? La primera idea fue usar una unión de es-





feras escaladas y situadas aleatoriamente dentro de un espacio cúbico. Quizá esto habría funcionado pero, temiendo que los recortes espaciales dejarán a los edificios con la apariencia de quesos de gruyere, se optó finalmente por emplear Heightfields. Estos objetos –como recordará el lector– se usan normalmente para crear mallas de apariencia montañosa y su aspecto irregular resultaba, pues, ideal para los recortes espaciales que se deseaban.

Así pues se creó una union de heightfields para efectuar las restas en cada edificio. Cada heightfiel fue primero rotado para que los picos de las montañas apuntaran en la dirección correcta. Hay un heightfield de recorte para cada fachada y también se emplea un quinto objeto de este tipo para efectuar recortes en el tejado. Los heightfields fueron escalados hasta alcanzar las dimensiones apropiadas y se usaron funciones rand en una serie de operaciones; al[illegible]s para introducir un valor de azar en la escala del heightfield en el eje de recorte, otras para randomizar la posición del objeto dentro de dicho eje y otras para rotarlo alrededor del eje de recorte. La finalidad de estas operaciones es, desde luego, procurar que cada edificio tenga un recorte espacial diferente al de los demás.

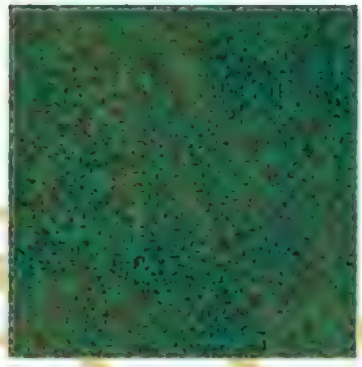
En cuanto a los bitmaps empleados para crear los heightfields, fueron generados según un procedimiento descrito brevemente en el Rendermanía número 1: creando primero los bitmaps de alzado con el propio POV y luego empleando dichos bitmaps para crear los objetos 3d. (Podéis ver un ejemplo adecuado de generación de un bitmap válido en imagen.pov). Las resoluciones de estos bitmaps son de 128*128 y 64*64. (En caso de que vayamos a crear escenas con muchos edificios, esta resolución debe ser tan baja como sea posible para ahorrar memoria). También se ha prescindido de la palabra smooth, ya que el suavizado del objeto de recorte no era necesario y evitándolo podíamos ahorrar un poco más de memoria.

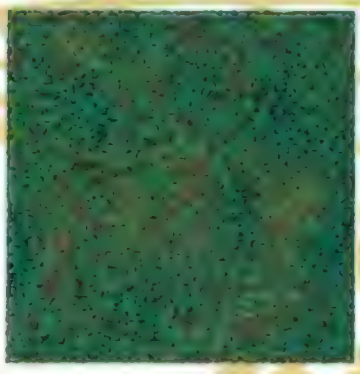
Finalmente –y para ahondar aún más en el aspecto ruinoso de nuestros edificios- podemos incluir un leve valor aleatorio de inclinación a cada edificio.

¿Macros o funciones?

En cuanto a la estructura de los ficheros de generación, en esta ocasión hemos empleado lo que algunos usuarios de POV han dado en llamar "funciones de usuario de POV". ¿Qué ocurre con estas "funciones"? Algunos usuarios –entre ellos vuestro seguro servidor– afirmaban que POV no las implementa mientras que otros aseguraban fervientemente que sí. Además en el número anterior, en el foro del lector, el grupo Mithrandor envió una carta tratando este particular. Después de leer dicha carta el autor de estas líneas se inclino a dar la razón a los "funcionalistas" si bien, una vez pasado el entusiasmo inicial, vuestro seguro servidor ha retomado nuevamente su opinión inicial, o sea, ¡no existen las funciones de usuario de POV!

¿Cómo debe entenderse esto? Cuando se habla de funciones de usuario de POV se está dando a entender que dichas funciones son similares en cuanto a su funcionamiento a las funciones de C. (De hecho el autor de estas líneas recuerda vagamente haber leído algún *faq* al respecto hace ya algún tiempo). Las funciones de C, como saben los programadores, son bloques de código que, generalmente, realizan tareas concretas cuyos objetivos puede variar dependiendo de los parámetros que se les pasen. Una vez compilado el programa, el código correspondiente a una función dada solamente se almacena una vez en memoria debiendo encargarse el programa de efectuar una llamada a la posición adecuada para que la función lleve a cabo su labor. Esta última condición es precisamente la que no cumplen las funciones de usuario de POV.





Veamos lo que sucede con la "función de usuario" incluida en edifi1.pvm:

1) POV está efectuando el "parsing" (compilación) de alguno de los ficheros .pov que llaman a edifi1.pvm. La "función" de edifi1.pvm creará un edificio cuyas dimensiones y forma dependerán de los valores que se pasen en los "parametros" nventanas, npisos, lateral, window, vigav y vigah. Después de dar un valor a cada parámetro preciso, la siguiente línea del fichero .pov será #include "edifi1.pvm".

2) Cuando el parser (la parte de POV encargada del parsing, o sea de convertir en "código 3d" los ficheros escénicos) llega a la línea del include, lo que sucede es, sencillamente, que se actúa como si todo el fichero edifi1.pvm estuviera escrito –a partir del include– en el archivo .pov en curso. POV "incluye" su texto línea a línea en el archivo .pov y cuando termina de parsear las líneas de edifi1.pvm continúa con las del archivo .pov inicial. Cuando, más adelante, en el mismo archivo, POV encuentre otra línea #include "edifi1.pvm", el proceso se repetirá, aunque puede que los resultados visuales sean muy diferentes ya que, presuntamente, los parámetros para esta llamada a la función serán diferentes a los de la primera llamada.

Exteriormente podrá parecernos que el funcionamiento de esta pov-función es similar al de las funciones de C pero, internamente, lo que sucede se parece mucho más al proceso de expansión de una macro de, por ejemplo, lenguaje ensamblador. (Por eso aquí empleamos la extensión pvm; pov-macro). Sin embargo, y precisiones aparte, el truco es de una gran utilidad ya que podremos ahorrarnos la escritura de muchas líneas y estructuraremos mejor nuestras escenas. El lector hallará ejemplos de todo esto en edificio.pov, ciudad.pov y portada.pov.

Los distintos tipos básicos de edificios

Las definiciones de los bloques básicos de los edificios se hallan en los ficheros pov, con lo que basta con cambiar dichos bloques para crear nuevos tipos. Los tipos que hemos empleado se hallan definidos dentro del switch, en ciudad.pov.

Escenas desastrosas

No hay duda de que, a pesar de que el fantasma de la guerra nuclear está un tanto difuminado hoy día, el genero catastrofista goza de buena salud. Multitud de autores se afanan en plasmar catástrofes de todo tipo, ya sea en letra impresa, en viñetas o en imágenes. Hoy nos sumaremos a esta ¿corriente? proponiendo una catástrofe mundial a escala global (y virtual afortunadamente), en la que el recalentamiento del planeta ha conducido al deshielo de las masas polares. ¿El resultado? Ciudades sumergidas y en ruinas como la que podéis ver en la portada.

Los pormenores de la construcción de los edificios de la portada ya han sido descritos en "Programación 3D". Ahora explicaremos el proceso por el cual la portada quedó tal como podéis verla.

Problemas en la creación del area catastrófica

Algunas de las ideas citadas en "Programación 3d" no han sido llevadas a cabo en nuestra portada. De hecho los edificios empleados en ella no son los más desastrosos (y resultones) de que disponemos, sino otros más sencillos. Los problemas han sido los de siempre: tiempo y, sobre todo, memoria. La escena de portada, por ejemplo, ha requerido unos 44 Megabytes. Esta cantidad, que parece excesiva ante la sencillez de los modelos, se explica por la existencia de los cientos de pequeños objetos simples que componen cada edificio. Estos objetos simples, rotados levemente en distintos ejes, producen (o deberían producir) una buena impresión de destrozo y de complejidad en cada modelo. En cuanto a la construcción de la ciudad, se ha empleado un bucle doble muy similar al que en su día se usó en las ciudades medievales (ya que el principio es el

mismo). Este bucle, en la portada, sirvió para generar 6*6 edificios. Estos "ruinosos" modelos, salvo por los suelos de cada piso, carecen de partes internas pero aún así –y asumiendo 2 vigas por cada ventana, la posible adición de un cristal y una media de 10*6 ventanas por fachada–, hay una media de medio millar de piezas simples por edificio.

Por otro lado, si en vez de los modelos empleados hubiéramos usado los edificios del 2110, la situación aún hubiera empeorado más, ya que cada edificio de esta época sufre recortes por parte de 5 heightfields.para aumentar la impresión de vejez. Además de esto, como siempre, mayor complejidad significa mayor lentitud en la generación de las pruebas. Quizá una posible solución hubiera estado en el empleo de uno u otro tipo de edificio atendiendo a la distancia de estos con la cámara.

Otro problema de la portada es la poca variación entre los distintos tipos de edificios. En la pov-macro switched.pvm sólo se definen 5 variaciones entre todos los casos posibles pero lo peor es que los distintos elementos simples usados para crear estas variaciones son demasiado parecidos entre sí. (Por esta razón no nos hemos molestado en añadir más casos). Las macros edifi0 y edifi1 emplean objetos declarados fuera (o sea en el fichero .pov de llamada) para preparar el montaje de cada edificio. Esto resulta cómodo pero poco ágil. Por otro lado quizá esto no sea necesariamente malo; el hacer tipos de edificios diferenciados entre sí probablemente daría como resultado que nos fijáramos aún más en las repeticiones, a menos, claro está, que escribiéramos una librería de edificios extensa. Esto no era lo que hoy se pretendía ¡ya que de todas formas los edificios iban a acabar como un montón de ruinas!

Un último punto a tener en cuenta es el hecho de que estamos usando una única semilla para decidir los factores aleatorios de todo: el número de ventanas y pisos, la forma de los edificios...,

etc. Esto significa que no podremos predecir el resultado visual final salvo de una manera muy aproximada. Claro que, por otro lado esto también tiene su atractivo, ya que bastará con cambiar el valor del parámetro de seed para obtener una ciudad distinta.

En portada.pov hemos utilizado el sistema del doble bucle con factores aleatorios para crear una malla de edificios. Después hemos colocado "a mano" un edificio de 22 pisos de altura y en él al monstruo. Puede suceder que un valor dado para seed dé como resultado unos edificios excepcionalmente altos en las filas más cercanas, con lo que no veremos nada más allá de ellos. También puede ocurrir que obtengamos una ciudad adecuada salvo por un detalle que no podremos cambiar ya que nuestro único control sobre el resultado será pedirle al programa que genere otra ciudad aleatoria.

La portada

Como antes se dijo, hemos incluido el mar para no tener que preocuparnos por los escombros, las calles, los coches, etc. Otra posible solución hubiera sido semienterrar la ciudad entre las dunas de un desierto. Esto, sin embargo, haría que fuese bastante mas difícil predecir la visibilidad de cada edificio concreto.

Aparte de esto se barajó la posibilidad de poner la cámara a ras del agua para enfocar algún edificio cercano en el que se viera a nuestro monstruo practicando alpinismo, pero esta idea fue desechada ya que hubiera sido muy difícil ofrecer así un panorama global de la ciudad. También estaba previsto (¡Ay!) emplear m[illegible] los de edificios del 2110 a los que se inclinaran sobre su eje según un factor aleatorio y también se acarició la idea de reescribir los edificios de modo que pudieran extenderse los pisos.

El modelo de Imagine

Estaba c[illegible] que había que incluir [illegible]gún modelo que diera una idea adecuada del tamaño de los edificios. La elección estaba muy clara dado que en el último número de Rendermania se incluyo un robot. Ahora bien, ¿cómo exportarlo a POV? Bien, en Rendermanía número 1 tratamos el tema de la exportación de [illegible]delos para POV. Todo cuanto se dijo en aquel artículo acerca la orientación de los ejes, el escalado de los modelos, etc. sigue siendo vigente. Pero, naturalmente, esto no nos servirá de mucho si no disponemos de alguna herramienta que "traduzca" el formato de Imagine a POV. El autor de estas líneas no ha hallado aún esta herramienta. Por esta razón el monstruo creado con Imagine fue grabado en formato DXF. Hecho esto las alternativas son algo más amplias ya que existen muchos programas que "entienden" este formato; Wcvt2pov puede, por ejemplo, leer DXF y grabar ficheros pov. Y también podemos grabar en formato 3ds y convertir luego a pov con 3ds2pov. Esperamos hallar algún buen conversor de objs de Imagine en un futuro.

POV 3.01

Finalmente hay que reseñar que todos los renders del presente número –salvo las páginas dedicadas a Imagine– fueron hechos con POV 3.01, una actualización que corrige varios bugs de la anterior y que podéis encontrar en la siguiente dirección de internet:

http://www.povray.org

Además, esta nueva versión permite extraer un doctorial en formato html y otro en modo texto, por si no nos hace gracia el modo de ayuda normal. Podemos extraer el fichero html y meterlo en un subdirectorio llamado html con la siguiente línea:

Phe2html -ipovhelp.phe -ohtml\
index.htm -ppov

También podemos extraer el fichero ascii con:

phe2txt -povhelp.phe

El nuevo manual ha cambiado algunas explicaciones y añadido ejemplos. En cuanto a POV, parece haber mejorado su tratamiento de errores. ¡Qué lo disfrutéis!

NOTA IMPORTANTE

Para hacer vuestras propias pruebas: Copiad los ficheros .pov, .inc y .pvm a un mismo directorio. Meted ahí también la textura texed.tga (la textura de las paredes) y, si vais a emplear edifi1.pvm, meted también los ficheros image o cread los vuestros propios. También puede resultar necesario realizar ligeros cambios en los ficheros .pvm dependiendo de cual sea el fichero .pov tratado.

Nota importante. Podéis remitirnos vuestros trabajos o consultas, bien por carta a la dirección que figura en la segunda página de Pcmanía, o vía e-mail a rendermania.pcmania@hobbypress.es

Piezas para juegos y batallas virtuales

Nuestro foro de hoy resultará especialmente interesante a todos los rendermaniacos por su diversidad. Incluye, entre otras cosas, una librería de piezas para crear juguetes Tente de modo virtual, unas cabezas extraterrestres modeladas con Metaballs, más pov-piezas virtuales para la anunciada batalla entre orcos y humanos, etc, etc. Damas y caballeros, pasen, vean y asómbrense.

En el número 4 de Rendermanía publicamos el carro de vapor creado por **Óscar Álvarez Díaz.** Este vistoso vehículo inspirado en el mundo de Warhammer fue el único modelo superviviente del disco enviado por su creador. Escarmentado por este desastre informático, Oscar ha vuelto a enviarnos su trabajo adjuntando esta vez copias de seguridad ¡lo cual ha sido una suerte ya que uno de los discos volvió a fallar! (Rendermaniacos, tomad nota: Enviad copias de seguridad y procurad que los discos vayan bien protegidos en el envío). En fin, por fin podemos contar hoy con la famosa vagoneta de ataque Snotling (o algo parecido) que Oscar diseño para ayudar a los orcos en su batalla virtual contra los humanos. Disponemos ya de varios piezas así que pronto podremos preparar alguna batallita, aunque si alguien se anima a preparar el altar de guerra Imperial o el cañón de salvas, mejor. Enhorabuena por estas estupendas máquinas, Oscar.

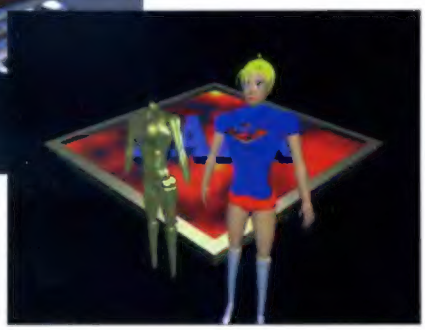
El Foro del Lector

Nasser Afkir Torres, miembro del grupo Njteam nos envía su última creación, una grúa basada en una pieza a escala. Puesto que Nasser pide una opinión diré que, desde luego, este trabajo representa un paso adelante (sobre todo en complejidad de modelado) con respecto al ordenador que enviaste la última vez. Lo único que no me ha convencido demasiado ha sido el fondo, que no me parece a la altura del modelo. En cuanto a los Ipas de 3D Studio que pides para el CD-Rom si no se trata de versiones shareware o autorizadas, no cuentes con ello. Respecto a lo que dices de los cuelgues de 3D Studio por falta de memoria, no me parece probable que sea éste el motivo. ¿Qué versión tienes?

Antonio José García Mora nos envía varias escenas con efectos atmosféricos y una utilidad para generar superficies extruidas y de revolución. Tomo nota de tu sugerencia acerca de dedicar un artículo al sistema de iluminación de POV. También a mi me gustaría que Marcos Fajardo se decidiera a pasar su trabajo a la última versión de POV. Es una pena que sus efectos se hayan quedado en la versión 2.2. En cuanto a lo que dices de los halos, aunque no tengo los ficheros originales (ya que fuiste cambiando los valores), creo que es posible que el problema resida en el escalado del halo con respecto al objeto contenedor, aunque eso no parece explicar la sombra. Por último, en lo que respecta al uso de Lparser, no recuerdo cómo dar colores individuales (sin trastear en el fuente con un editor). Es posible que dentro de poco veamos aquí algunas técnicas para la generación de árboles, ya que parece haber bastante gente interesada en el tema. Y para acabar, no he podido por ahora echar un vistazo a tu Editor de Splines. Pero, aunque no pretendo desanimarte, sera difícil que tu utilidad tenga mucha salida. A fin de cuentas este tema esta contemplado en Moray y otros programas.

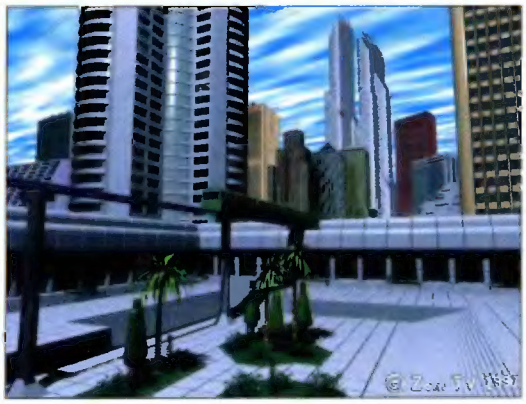
De inexcusable lectura es la interesantísima carta remitida por **David Lozano Lucas** donde explica como empleó el Metareyes para construir las Super Cabezas Extraterrestres que podéis ver aquí. Estas cabezas fueron creadas por David primero en plastilina y luego... pero lo mejor será que leáis su carta en el Foro. David incluye también una animación que mezcla animales reales con un fondo de 3D Studio (y que yo no he conseguido ver bien). Enhorabuena por tu original y excelente trabajo.

Daniel Susin Acebo también ha decidido tomar parte en la anunciada batalla virtual entre orcos y humanos y contribuye con este magnífica pov-catapulta que pasa a formar parte del bando orco. Daniel pide una crítica "constructiva" (¡Ja!) de sus imágenes. Bien ¿qué podemos decir? La catapulta está esplendidamente diseñada y montada, las texturas (a pesar de tus temores) quedan perfectamente y no encuentro ningún punto débil salvo en la bola de fuego que, por otro lado, no es fácil de hacer. Quizá hubieras debido poner una piedra normal. Anótate un 10.5 sobre 10. En cuanto a lo que preguntas sobre exportación de modelos de Imagine para POV, supongo que habrás leído el resto del presente número.



José María Tejeda Martín -Dark infográfica- es otro infografista claramente inspirado en el ejemplo de Tomwoof y sus mangababes. José María ha creado un modelo femenino para 3D Studio al que llama Chunli (pero que no tiene nada que ver con el personaje de Street Fighter que pega palizas al autor de estas líneas los sábados por la mañana). Este modelo, del que podéis ver unos ejemplos aquí, puede ser recibido por todo aquel que decida registrarse (ved los detalles explicados por Joée María en los ficheros del foro). Buena suerte para ti y para los siguientes modelos de la librería Chunli.

El Foro del Lector



Zeni Tallò nos envía desde Sabadell cuatro escenas magníficas. En su carta -que Zeni no ha enviado como fichero- este autor declara que aún se considera un novato en el uso de POV y que desconoce el funcionamiento de las estructuras complejas del programa tales como Smooth Triangle (¿?) En su carta Zeni solicita una crítica constructiva y pregunta por el grado de calidad de sus trabajos. Finalmente Zeni pregunta por qué es tan difícil modelar formas orgánicas con POV; el robot femenino iba en principio para chica de carne y hueso.

Bien, vayamos por partes: Amigo Zeni tus escenas son muy buenas. sobre todo en el aspecto del modelado, y me han gustado mucho. Sin embargo, has omitido el envío de los ficheros de generación sin los cuales mi crítica tiene que reducirse necesariamente a un breve comentario acerca de lo resultonas que son tus imágenes. De éstas me gustó sobre todo la ciudady también me gustó mucho el androide femenino, aunque hubiera preferido verlo de cuerpo entero. Sin embargo, me veo obligado a señalarte que, al no haberme remitido los ficheros de generación. no puedo asegurar públicamente que las imágenes sean tuyas. Si eres un lector reciente quiza no hayas leído mis reiteradas explicaciones acerca del porqué de todo esto. Quiero aprovechar la ocasión para pedir a todos que enviéis alguna prueba de que las escenas son efectivamente vuestras. Así, a partir de ahora: 1) No hace falta que enviéis los archivos de generación completos. Bastará con que, al renderizarlos, se creen escenas con elementos significativos de las imágenes enviadas. Estos archivos, en cualquier caso, no serán publicados si el autor así lo declara. 2) Si se trata de escenas de 3D Studio, Imagine o de cualquier otro programa de tipo poligonal, se agradecera una pequeña explicación acerca de los modelos o de cualquier otra cosa que pueda permitirnos confiar en la autoría de las obras enviadas. No es preciso que enviéis las mallas si no deseáis verlas publicadas. 3) Si es posible, enviad los discos por duplicado. 4) Si me entero de que alguien me "cuela" alguna obra robada seré el primero en denunciar el hecho y poner pasquines hasta en el himalaya para que el culpable no escape al escarnio público.

Cambiando de tema, nadie usa manualmente Smooth Triangle, ya que es una primitiva puesta ahí por los desarrolladores de POV para que sea usada por los programas de utilidad que convierten mallas de otros programas; para que sus modelos puedan generarse en POV, claro. En cuanto a lo de las formas orgánicas, no se trata de una falta de conocimientos o de habilidad por tu parte, sino de un campo muy, muy difícil, incluso empleando los modeladores y las utilidades más poderosas.

Jorge Martín Cuervo se lleva hoy la medalla a la originalidad puesto que nos envía una colección de piezas de Tente diseñadas conjuntamente con POV y 3D Studio. Con dichas piezas, Jorge ha montado los barcos de juguete que podemos ver aquí. (¡!). Jorge también nos envía un proyecto de cocina creado con 3D Studio y que ha quedado incompleto debido a otro de esos desastres informáticos. Jorge pide que enviémos opiniones sobre su trabajo así que aprovecharé para dar la mía: La idea de la colección de piezas Tente me parece muy original y bien realizada pero... ¿Se puede construir un castillo con ello?